QoS Composition and Analysis in Reconfigurable Web Services Choreographies

Ajay Kattepur, Nikolaos Georgantas, Valérie Issarny

QoS Composition and Analysis in Reconfigurable Web Services Choreographies

Ajay Kattepur, Nikolaos Georgantas & Valérie Issarny
Equipe ARLES, INRIA Paris-Rocquencourt
Domaine de Voluceau, Le Chesnay, France.
Email: firstname.lastname@inria.fr

***Abstract*—Quality of Service (QoS) in *orchestrated* web services compositions have been well studied with probabilistic and multi-dimensional models. *Choreographies* that involve message passing among services, on the other hand, require further analysis. In this paper, we begin with the set of QoS domains that may be studied in case of choreographies and the algebraic rules for their composition. As choreographies manage QoS composition in a distributed fashion, techniques to enrich functional specifications with QoS are examined using the model proposed in the *CHOReOS* project. These are further analyzed with choreographies that may reconfigure due to functional or QoS requirements. Studies on the effects of such reconfiguration on multiple QoS domains can lead to better understanding of optimal runtime configurations along with associated tradeoffs. A goal programming approach is also proposed to choose *Pareto optimal* solutions with respect to diverse QoS domains.**

***Keywords*-QoS; Service Choreographies; Reconfiguration; Goal Programming.**

I. Introduction

Web services and associated Quality of Service (QoS) have received considerable attention in the research community relating to discovery, late-binding and optimal end-to-end compositions [1], [2]. Maintaining efficient QoS levels of invoked services is a major prerogative of composite web service orchestrations, with service level agreements (SLAs) [3] being employed to ensure this. QoS metrics being multi-dimensional random variables, the treatment of QoS composition and SLAs tends toward probabilistic criterion [4].

Web services choreographies involve a "global view" of message passing and coordination among various sets of services. Unlike orchestrations that have a centralized control flow and may be described/run on engines such as BPEL [5], choreographies involve a shared service composition framework where only the participants' functionality and associated message passing are described [6]. Languages developed to describe such message passing and coordination within choreographies, such as BPMN [7], WS-CDL [8] and BPEL4Chor [9] have the following elements: (a) *Messages*: types and content of messages passed between participants (b) *Message ordering*: time-constraints for asynchronous/synchronous messages with associated pre-/post- conditions (c) *End points*: choreography end points that may be accessed with specific protocols.

When dealing with QoS of web services, techniques specified by Zeng et al. [2] may be applied to the domains such as *latency*, *availability*, *security* and so on (see [1] for a survey). For choreographies, in addition to these metrics, the evaluation must also take into account the integrity of messages passed. Unlike data independent orchestrations, where the centralized orchestrator may separate functional and QoS analysis, choreography QoS have to consider [10]:

- *Deadlock Freeness*: A deadlock occurs when the runtime deployment of a choreography reaches a (non-final) state that cannot be left without violating the message ordering.
- *Conformance*: A participant of a choreography conforms to a message passing specification dependent on aspects such as correctness of specification.
- *Realizability*: Choreographies may have participants that can generate adverse side-effects to other participants due to non-compliance of specifications/enforcement policies.

While these functional properties are, in general, verified at *design-time*, they involve assumptions on *runtime* QoS behavior. Deadlock freeness assumes constant availability (response produced within timeout) while realizability assumes security/data integrity of messages. Thus, there are intricate links between choreography functioning and analysis of QoS metrics.

In this paper, we build on the algebraic formulation proposed in [4] to integrate QoS metrics for service choreographies. Operators to compose functional behavior such as *message integrity* and *accuracy* of the choreography are studied, as they influence (and are influenced by) QoS. As the management of QoS in choreographies needs to be decentralized, choreography operations are enhanced with QoS increments. For this, we extend the BPMN-based choreography model described in the *CHOReOS* project [11] (http://www.choreos.eu) with QoS values to propagate and aggregate these metrics.

Analysis is then performed with cases where the choreography adapts to runtime requirements. Such

This work has been partially supported by the European Union's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (project CHOReOS).

configuration-aware/self-healing choreographies that dynamically adapt [12] may be triggered by functional, behavioral changes in services (non-availability for example) or QoS requirements that differ from design-time assumptions. Aspects such as deadlock freeness and message ordering, that are taken care of during design time, have to consider such configurations. Studying the QoS of participating services in such cases further takes into account the multi-dimensional, probabilistic QoS with added variability due to such adaptation. By doing so, extended studies on "good" and "bad" configurations may be done, dependent on changes produced in composite choreography QoS. As there are a number of tradeoffs typically involved in reconfigurations, accurate study of the QoS composition process along with message passing is crucial to ensure an "optimal" functional+QoS configuration.

We emphasize three principal contributions of our work:

1) An algebraic formulation that can handle multi-dimensional QoS metrics and corresponding composition for service choreographies.
2) Study of the effect of runtime reconfiguration on the composition of QoS metrics and assumptions taken at design-time.
3) A *Pareto optimal* technique to manage *goal*-dependent selection among traded-off QoS configurations.

The rest of the paper is organized as follows: Section II introduces QoS domains of interest and the algebraic operators for composing them. The process of integrating these metrics within the choreography specifications are studied in Section III. Management of these metrics in terms of contracts and reconfiguration are briefly analyzed in Section IV. A Pareto optimal policy for selecting among various alternative configurations is analyzed in Section V. In Section VI we provide an *airport check-in* example of reconfigurable choreographies with a detailed analysis of QoS composition and tradeoff. This is followed by related work and conclusions in Sections VII and VIII, respectively.

II. QoS in Choreographies

In this section, we review the domains of QoS relevant to our analysis and the algebra used to compose them.

A. QoS Domains

From an atomic/composite web service perspective, QoS properties of interest may be represented by multi-dimensional [1] and probabilistic [4] domains:

- *Latency* δ: End-to-end response time between web service invocation and response. *Availability* may be subsumed as responding within a particular timeout period. Typically represented by a heavy tailed distribution.
- *Throughput* λ: Number of times a web service may be accessed in a particular period. Represented by an exponentially valued random variable.
- *Cost* $\mathcal{C}$: Amount a client has to pay for the service (per-invocation, subscription policies). May be drawn from a uniform/normal distribution with items accumulated.
- *Data Quality* Δ: The quality of data produced by the service including freshness, accuracy and correctness. Represented as a binomial random variate with values *(high, low)*.
- *Security* $\mathcal{S}$: Confidentiality of the data produced and transmitted by a web service. Represented as a binomial random variate with values *(high, low)*.

The above metrics may be extended with domains from choreography behavior, which relate both functional and QoS properties [10]:

- *Accuracy* α: Measures how accurately the choreography function follows its specification. Represented as binomial random variate values *(valid, invalid)*.
- *Message integrity* $\mathcal{M}_i$: Capability of the choreography to prevent data corruption and unauthorized access to information during choreography enactment. Represented as binomial random variate values *(high, low)*.
- *Messaging Efficiency* $\mathcal{M}_e$: This captures the ordering and content of messages being exchanged by the choreography. Aspects such as deadlock freeness must be taken into account. Represented as binomial random variate values *(high, low)*.

It must be noted here that these domains are by no means exhaustive but represent a cross-section of possible QoS that may be applied.

B. QoS Algebraic Composition

As specified by Rosario et al. [4], a *QoS metric* q is a tuple:

$$q = (\mathbb{D}, \leq, \oplus, \bigwedge, \bigvee) \tag{1}$$

1) $(\mathbb{D}, \leq)$ is a QoS domain with corresponding partially ordered set of QoS values
2) $\oplus : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ defines how QoS gets incremented by each new event/operation. It satisfies the following conditions:
   - $\oplus$ possesses a *neutral element* 0 satisfying $\forall q \in \mathbb{D} \Rightarrow q \oplus 0 = 0 \oplus q = q$.
   - $\oplus$ is *monotonic*: $q_1 \leq q_1'$ and $q_2 \leq q_2'$ imply $(q_1 \oplus q_2) \leq (q_1' \oplus q_2')$.
3) $(\bigwedge, \bigvee)$ represent the lower and upper lattice, meaning that any $q \subseteq \mathbb{D}$ has a unique least lower, upper bound $(\bigwedge_q, \bigvee_q)$. When taking the *best* response with respect to the ordering $\leq$, the lowest QoS is taken with $\bigwedge$. When synchronizing events, the operator $\bigvee$ amounts to taking the *worst* QoS as per the ordering $\leq$.

Basic classes of QoS domains are displayed in Table I. Domains $\mathbb{D}$ may be real numbers $\mathbb{R}_+$ as for latency, generic domains $\mathbb{Q}$ as for security *(high, low)* or mapped

QoS Metric		$\mathbb{D}$	$\leq$	$\oplus$	$\bigwedge$	$\bigvee$
δ:	Latency	$\mathbb{R}_+$	$\leq$	$+$	min	max
$\mathcal{C}$:	Cost	$\mathbb{Q} \mapsto \mathbb{N}$	$\subseteq$ on $\mathbb{N}$	$\bigcup$	$\bigcap$	$\bigcup$
$\mathcal{S}$:	Security	$\mathbb{Q}$	$\geq$ on $\mathbb{Q}$	$\bigvee$	max	min
Δ:	Data Quality	$\mathbb{Q}$	$\geq$ on $\mathbb{Q}$	$\bigvee$	max	min
λ:	Throughput	$\mathbb{R}_+$	$\leq$	$+$	min	max
α:	Accuracy	$\mathbb{Q}$	$\geq$ on $\mathbb{Q}$	$\bigvee$	max	min
$\mathcal{M}_i$:	Msg. Integrity	$\mathbb{Q}$	$\geq$ on $\mathbb{Q}$	$\bigvee$	max	min
$\mathcal{M}_e$:	Msg. Efficiency	$\mathbb{Q}$	$\geq$ on $\mathbb{Q}$	$\bigvee$	max	min

Table I
QoS Domains and Algebra.

BPMN Workflow Pattern	Operator
Sequence	$\oplus$
Exclusive Choice - Simple Merge	$\bigwedge$
Multi-Choice - Multiple Merge	$\bigwedge$
Parallel Split - Synchronization/Discriminator	$\bigvee$
N out of M Join; Cycles/Loops	$\bigvee$
Milestone; Cancel Activity/Case; Termination	$\bigvee_\perp$

Table II
QoS Composition with Business Process Patterns.

to sets of values $\mathbb{Q} \mapsto \mathbb{N}$ as with cost. The partial ordering can be "less is better"($\leq$ with latency), "more is better"($\geq$ with security) or using subsets($\subseteq$ with cost). The aggregation operator $\oplus$ results in addition for the case of latency; however for domains such as cost(Union $\bigcup$) or security($\bigvee$), this would mean taking the supremum of the QoS values aggregated.

We assume that both choreographies and orchestrations (as previously studied in [4]) have similar composition models for end-to-end QoS with additional aggregation required (Section III). Some metrics such as α, $\mathcal{M}_i$, $\mathcal{M}_e$ along with $\mathcal{S}$, Δ are closely dependent on the "net" effect of the choreography's functional performance. For example, the loss of *message efficiency* $\mathcal{M}_e$ in any of the choreography participants can lead to deadlock ($\bigvee \mathcal{M}_e$) of the entire choreography. Finite domains such as {high, medium, low} can be composed with $\bigvee$ of the result of a composition (high security information plugged to a low security source).

In order to briefly discuss this composition process, we present Table II, where common BPMN [7] workflow patterns are analyzed with their corresponding QoS operators. The use of the sequential aggregation operator $\oplus$ and the choice-based infimum $\bigwedge$ follows from our algebra. For parallel invocations with synchronizations and looping, the supremum $\bigvee$ should be used. We also specify $\bigvee_\perp$ for terminations/cancellations of executions (timeouts or sunk-costs, for instance). Such aggregation techniques are evaluated in detail in [13].

Encoding the QoS algebra (Table I) can be done in a variety of scripting languages. Such a specification may be invoked by any choreography description language to incorporate QoS composition. It specifies how to calculate the QoS increments by employing the associated algebra with domains $\mathbb{D}$, partial order $\leq$ and operations $(\oplus, \bigvee, \bigwedge)$. In case of orchestrations, the increments and composition may be managed by the orchestrator itself (for monitoring SLAs, for instance). In case of choreographies, this has to be done in a distributed manner with every functional operation being tagged with a QoS increment (for aggregation/composition with rules of Table II). An example specification in *python* is shown below:

```
class Latency:                          # Latency
    def __init__(self):
        self.QoS = student_t(nu)        # student-t dist.
    def QoSOplus(self,q2):              # oplus operator
        return (self.QoS + q2)
    def QoSCompare(self,q2):            # partial order
        return self.QoS <= q2
    def QoSInfimum(self,q2):            # infimum operator
        v = [self.QoS,q2]
        return v.sort()
    def QoSSupremum(self,q2):           # supremum operator
        return max(self.QoS,q2)
```

III. QoS within Choreography Specifications

The choreography model used is that of the *CHOReOS* project [11], which focuses on large scale choreographies that connect heterogeneous, adaptable and QoS-aware services [11]. The generic representation of the *CHOReOS* model is shown in Fig. 1 with *services* represented by a generic *interface* to abstract behavior. The functional interface of the choreography specification provides a description of the *operations*, *messages* and *constraints*. Note that there are three types of services that may participate:

1) *Stateless Services*: These are just treated atomically with service requests-responses.
2) *Stateful Services*: Have internal transitions that are concealed by the interface. They also have externally observable transitions that should conform with the message passing of the choreography, which are exposed by the interface.
3) *Composite Services*: Modeled as orchestrations that can have varied internal control flow. Note that the interface may hide the internal behavior and these may be considered a distributed implementation of a single *stateful* service.

At the choreography level in Fig. 1, specific *roles* that need to be performed by the choreography are assigned to concrete services. The *coordination delegates* are special entities that wrap such services and adapt - if necessary - their behaviors to their assigned roles, i.e. to the choreography message passing specifications. At the level of the underlying middleware-based communication, the *generic application connectors* ensure interoperability across services that may employ different interaction paradigms, e.g., publish-subscribe and tuple spaces besides the typical client-server Web service interaction.

While *CHOReOS* choreographies are specified in BPMN [7], this is formally abstracted with labeled transition system (LTS) specifications, as follows [11]:

Definition 1. *Service LTS $L = (S, T, D, S_0)$. S is a finite set of states, $T \subseteq O$ is a finite set of transitions, where O is the set of service operations that the service may provide (inbound invocations) or require from other*

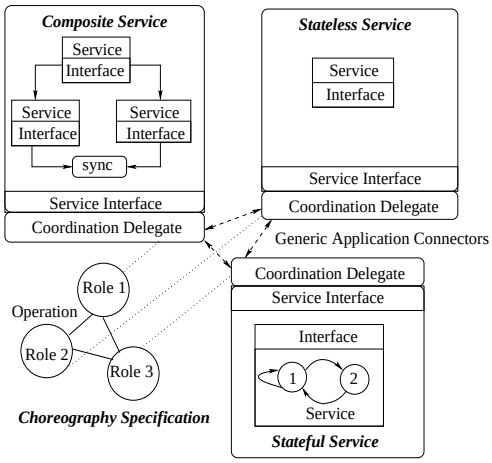


Figure 1. *CHOReOS* Choreography Model [11].

services (outbound invocations) as defined by its functional interface, D is the transition relation $S \times T \times S$ and S_0 is the initial state.

Definition 2. *Choreography LTS $L_c = (l_c, Roles)$ with LTS l_c having the transition relation $T \subseteq Roles \times O \times Roles$, with Roles being the universal set of the local roles provided by the choreographies' service participants and O being the universal set of the roles' operations.*

This definition of a Choreography LTS can be used to bind roles to concrete services by projecting the *Choreography LTS* onto each *role LTS* and by matching each role LTS with a concrete service LTS. We extend the above definition to a QoS-aware choreography by associating QoS increments with specific operations, as follows:

Definition 3. *QoS-centered Choreography LTS $L_c(Q) = (l_c, Roles, Q)$ with LTS l_c having the transition relation $T \subseteq Roles \times O_Q \times Roles$, with O_Q being the QoS increments from domain $\mathbb{D}_Q$ associated with operation O.*

Definitions 2 & 3 produce the ability to generate the output tuple of *(functional output, QoS increment)*, where the QoS increment is associated to the service operation that produces the functional output. Such an output can be used by choreography participants to monitor QoS increments in a distributed fashion, which differs from central monitoring of orchestrations. In the next section, we study some of these QoS management policies.

IV. QoS Management

The algebraic formulation presented in Tables I and II along with the QoS interpretation of Definition 3 can be used in a variety of ways for managing choreographies. These interactions are managed either through contracts or by reconfiguring at runtime. Interactions that need to be strictly monitored for QoS are provided contracts while best-effort QoS interactions are dealt with through runtime reconfigurations.

A. QoS Contracts

Service Level Agreements [3] make use contractual guarantees for strict monitoring of composite services' performance. The procedure outlined in Tables I & II are combined with analytic (queues, flows) or simulation techniques (e.g. [14]) to generate end-to-end performance metrics. While the end-to-end performance of a choreography is generally not contractually guaranteed, participating services can have contractual guarantees. Contracts are applied to service interactions in choreographies that need to strictly adhere to requirements/QoS expectations, in order to prevent deviations in functionality.

B. Reconfiguration

Reconfiguration from a choreography perspective means either replacing certain services at runtime (due to change in requirements, QoS) or modifying the composition. This is done via self-healing systems, model-based techniques or goal driven multi-stage approaches, for instance [12]. Further analysis of end-to-end participants' QoS should be analyzed before/after the re-configuration has occurred. Reconfiguration assumes that runtime behavior may vary and has services/strategies to adapt when such behavior occurs.

Algorithm 1: Runtime QoS-Aware Reconfiguration

```
1  Set Functional, QoS Goals: 𝔾[f], 𝔾[q]
2  Set Design-time Configuration: C_i ∈ Composition ℂ; i := 1
3  while i ≤ i_max do
4      Deploy Runtime Configuration C_i
5      Check Goals 𝔾[f_{C_i}], 𝔾[q_{C_i}] achieved by C_i
6      if (𝔾[f_{C_i}] ≤ 𝔾[f]) ∧ (𝔾[q_{C_i}] ≤ 𝔾[q]) then
7          Set C_out := C_i
8          break
9      else
10         q := ⋁ q;   ∀q ∈ 𝔻(δ | λ | C)
11         q := q_∅;   ∀q ∈ 𝔻(S | Δ | α | M_i | M_e)
12         i := i + 1
13         Re-Configure C_i ∈ ℂ
14 Return QoS-optimal C_out
```

We make use of Algorithm 1, that represents a methodology for runtime reconfiguration when design-time QoS objectives are not met. Loosely, we refer to a *configuration* $\mathbf{C}_i$ to be a runtime instance of a service composition $\mathbb{C}$. If a particular configuration that is designed $(\mathbb{G}[f_{\mathbf{C}_i}], \mathbb{G}[q_{\mathbf{C}_i}])$ cannot meet the functional+QoS goals $(\mathbb{G}[f], \mathbb{G}[q])$ represented with partial ordering $\leq$, it may trigger a reconfiguration procedure specified by modeling/agent based techniques [12]. Note that we have a *reset QoS* step $q := \bigvee q/\ q_\emptyset$ when the objectives are not met; this step appends the supremum $\bigvee q$ of domains such as latency δ/cost $\mathcal{C}$ that have already been exhausted by the configuration run; it resets the null value $q_\emptyset$ to domains such as accuracy α/security level $\mathcal{S}$. Algorithm 1 is computationally inexpensive (maximum iterations $i_{\max}$) and checks if a particular configuration reaches the functional+QoS goals of the choreography.

An alternative, more computationally expensive technique, is to search the entire configuration space for the optimal configuration with respect to goals, which is given in Section V. Such techniques of searching through large composition spaces has been studied further in [15], using the compositional decision-making process.

V. QoS Optimal Configurations

QoS being multi-dimensional in nature, the optimal binding/selection procedure would generally be modeled as a multi-objective optimization problem. There has been work done to model this problem from the web services selection/composition perspective as multidimensional multiple-choice knapsack problem (MMKP) [16], integer/linear programming [2], [17] or with analytic hierarchy process weights [18].

In Table I and Algorithm 1, we introduce multiple metrics that are formulated as individual objective values. As these metrics have dependencies and tradeoffs, a single solution that simultaneously optimizes each objective is not trivial. Typically, evaluation would mean studying the *Pareto Optimality* of solutions:

Definition 4. *For a multi-objective function* $\mathbf{F}(\mathbf{q}) = [\mathbf{F}(\mathbf{q})_1, \mathbf{F}(\mathbf{q})_2, \dots \mathbf{F}(\mathbf{q})_k]^T$ *to be minimized in domain* $\mathbb{Q}$*, a point* $\mathbf{q}' \in \mathbb{Q}$ *is* Pareto Optimal *iff there does not exist another point* $\mathbf{q} \in \mathbb{Q}$ *such that* $\mathbf{F}(\mathbf{q}) \leq \mathbf{F}(\mathbf{q}')$ *and* $\mathbf{F}(\mathbf{q})_i < \mathbf{F}(\mathbf{q}')_i$ *for at least one objective function.*

A point is Pareto optimal if there is no other point that improves at least one objective function without detriment to another function. We consider using a goal programming approach [19] to study the optimality of the outputs as set out in Algorithm 1. By specifying goals $\mathbb{G}[\mathbf{q}]_j$ for each objective function $\mathbf{F}(\mathbf{q})_j$, it is possible to study the deviation of goals produced due to each reconfiguration. The total deviation from goals $\sum_{j=1}^{k} \left(d(\mathbf{q})_j^+ + d(\mathbf{q})_j^-\right)$ is to be minimized with $d(\mathbf{q})_j^+ \geq 0$ (overachievement) and $d(\mathbf{q})_j^- \leq 0$ (underachievement) from goal $\mathbb{G}[\mathbf{q}]_j$ for objective $\mathbf{F}(\mathbf{q})_j$. The optimization problem:

$$\begin{array}{ll} \min_{\mathbf{q}\in\mathbf{Q}}: & \sum_{j=1}^{k} \left(d(\mathbf{q})_j^+ + d(\mathbf{q})_j^-\right) \\ \text{Subject to:} & \mathbf{F}(\mathbf{q})_j + d(\mathbf{q})_j^+ - d(\mathbf{q})_j^- = \mathbb{G}[\mathbf{q}]_j \\ & d(\mathbf{q})_j^+,\ d(\mathbf{q})_j^- \geq 0; \quad j = 1, 2, \dots, k \end{array} \tag{2}$$

This may be instantiated to multiple domains (Table I) with the intention of locating the Pareto optimal QoS configuration:

Minimize:

$$\sigma^2(\hat{\delta})^- + \sigma^2(\hat{\delta})^+ + d(\hat{\mathcal{C}})^- + d(\hat{\mathcal{C}})^+ + \sigma^2(\hat{\lambda})^- + \sigma^2(\hat{\lambda})^+ + d(\hat{\Delta})^- + d(\hat{\Delta})^+ + d(\hat{\mathcal{S}})^- + d(\hat{\mathcal{S}})^+ + d(\hat{\alpha})^- + d(\hat{\alpha})^+ + d(\hat{\mathcal{M}}_i)^- + d(\hat{\mathcal{M}}_i)^+ + d(\hat{\mathcal{M}}_e)^- + d(\hat{\mathcal{M}}_e)^+$$

Subject to:

$$\mu(\hat{\delta}) + \sigma^2(\hat{\delta})^+ - \sigma^2(\hat{\delta})^- = \mathbb{G}[\hat{\delta}]; \quad \sigma^2(\hat{\delta})^+, \sigma^2(\hat{\delta})^- \geq 0$$
$$\bigvee(\hat{\mathcal{C}}) + d(\hat{\mathcal{C}})^+ - d(\hat{\mathcal{C}})^- = \mathbb{G}[\hat{\mathcal{C}}]; \quad d(\hat{\mathcal{C}})^+, d(\hat{\mathcal{C}})^- \geq 0$$
$$\mu(\hat{\lambda}) + \sigma^2(\hat{\lambda})^+ - \sigma^2(\hat{\lambda})^- = \mathbb{G}[\hat{\lambda}]; \quad \sigma^2(\hat{\lambda})^+, \sigma^2(\hat{\lambda})^- \geq 0$$
$$\bigvee(\hat{\Delta}) + d(\hat{\Delta})^+ - d(\hat{\Delta})^- = \mathbb{G}[\hat{\Delta}]; \quad d(\hat{\Delta})^+, d(\hat{\Delta})^- \geq 0$$
$$\bigvee(\hat{\mathcal{S}}) + d(\hat{\mathcal{S}})^+ - d(\hat{\mathcal{S}})^- = \mathbb{G}[\hat{\mathcal{S}}]; \quad d(\hat{\mathcal{S}})^+, d(\hat{\mathcal{S}})^- \geq 0$$
$$\bigvee(\hat{\alpha}) + d(\hat{\alpha})^+ - d(\hat{\alpha})^- = \mathbb{G}[\hat{\alpha}]; \quad d(\hat{\alpha})^+, d(\hat{\alpha})^- \geq 0$$
$$\bigvee(\hat{\mathcal{M}}_i) + d(\hat{\mathcal{M}}_i)^+ - d(\hat{\mathcal{M}}_i)^- = \mathbb{G}[\hat{\mathcal{M}}_i]; \quad d(\hat{\mathcal{M}}_i)^+, d(\hat{\mathcal{M}}_i)^- \geq 0$$
$$\bigvee(\hat{\mathcal{M}}_e) + d(\hat{\mathcal{M}}_e)^+ - d(\hat{\mathcal{M}}_e)^- = \mathbb{G}[\hat{\mathcal{M}}_e]; \quad d(\hat{\mathcal{M}}_e)^+, d(\hat{\mathcal{M}}_e)^- \geq 0$$
$$\left(\mathbb{G}[\hat{\delta}] + \mathbb{G}[\hat{\Delta}] + \mathbb{G}[\hat{\mathcal{S}}] + \mathbb{G}[\hat{\alpha}] + \mathbb{G}[\hat{\mathcal{M}}_i] + \mathbb{G}[\hat{\mathcal{M}}_e]\right) \times \left(\mathbb{G}[\hat{\mathcal{C}}] + \mathbb{G}[\hat{\lambda}]\right) = \mathcal{K}$$

This is an instantiation of eq. 2 with the objective function minimizing the sum of deviations $d(\hat{\mathbf{q}})^+, d(\hat{\mathbf{q}})^-$, where $\hat{\mathbf{q}}$ represents the values in domain $\mathbb{Q}$ normalized to be in range $[0, 1]$ (like in Zeng et al. [2]). The constraints also follow from eq. 2 with values $\mathbf{F}(\mathbf{q})$ and goals $\mathbb{G}[\mathbf{q}]$ instantiated from QoS in Table I. For metrics such as latency δ, the mean $\mathbf{F}(\hat{\mathbf{q}}) = \mu$ and variance $d(\hat{\mathbf{q}}) = \sigma^2$ is used to set the quality constraints. For other metrics such as data quality Δ and message integrity $\mathcal{M}_i$, the supremum $\mathbf{F}(\hat{\mathbf{q}}) = \bigvee$ with deviations $d(\hat{\mathbf{q}})^+, d(\hat{\mathbf{q}})^-$. The tradeoff constraints are included with improvement in some goals meaning deterioration in others. This is represented with the goals for cost and throughput $\left(\mathbb{G}[\hat{\mathcal{C}}] + \mathbb{G}[\hat{\lambda}]\right)$ inversely proportional to the sum of the goals for the other metrics $\left(\mathbb{G}[\hat{\delta}] + \mathbb{G}[\hat{\Delta}] + \mathbb{G}[\hat{\mathcal{S}}] + \mathbb{G}[\hat{\alpha}] + \mathbb{G}[\hat{\mathcal{M}}_i] + \mathbb{G}[\hat{\mathcal{M}}_e]\right)$, via linear constant $\mathcal{K}$. Essentially, this states that the deterioration in cost (higher) and throughput (lower number of invocations) improves domains such as latency (lower) and security (better).

While the computational complexity of the approach is greater than using linear/weighted optimization, the quality of the output is superior as it uses only the partial orders along with the probabilistic nature of these domains. Such an improvement may also be seen in stochastic programming dependent composition approaches like in [20]. The Pareto optimal nature of this output captures both the effect of functional changes in choreographies (deadlock, conformance through $\mathcal{M}_i, \alpha$) while also monitoring service behavior though metrics such as δ, λ. This is crucial for choreographies where improvement in functionality may be traded-off with goal deviation in certain QoS domains. Through this, the optimization formulation loosely resembles the actual choreography performance – where metrics are not independent of each other.

VI. Reconfigurable Choreography Example

In order to study the effect of reconfiguration in QoS-dependent choreographies, we make use of the motivating *airport check-in* example.

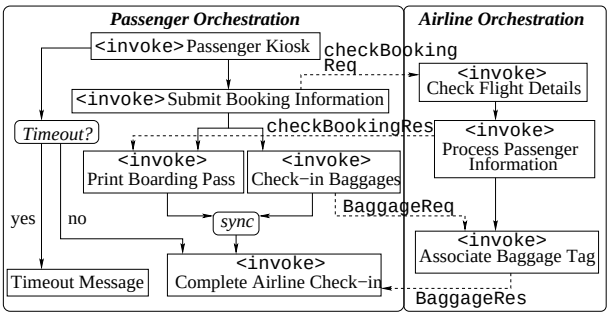
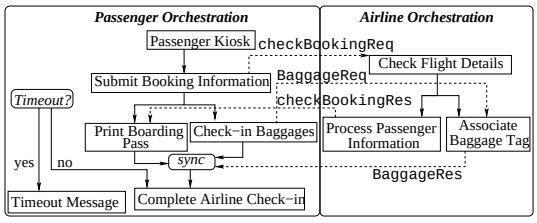


Figure 2. The design-time *airport check-in* choreography consisting of the *passenger* and *airline* orchestrations with message passing represented with dotted lines.

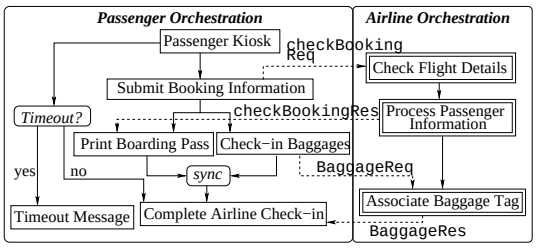
A. Example Description

In Fig. 2, the *passenger* and *airline* orchestrations participate with a series of messages to complete the choreography implementation. This is represented in a generic form, which may be translated to BPMN [7]/WS-CDL [8]/BPEL4Chor [9]. In this example, we consider two composite services (modeled as orchestrations of stateless services, but could have equally been two atomic stateful services)that pass messages specified in the choreography. The *Passenger orchestration* relays the passenger's request to the `passenger kiosk` service that subsequently invokes the `submit booking information` service. This then passes to the `print boarding pass` and the `check-in baggages` services in parallel with data collated at the `complete airline check-in` service. In this orchestration, *function* and *QoS* interact, with the `timeout` service monitoring latency (response invalid if after timeout). Similarly, the *Airline orchestration* consists of `check flight details`, `process passenger information` and `associate baggage tag` services in sequence. In order for the *airport check-in* choreography to function properly, messages (represented by dotted lines) need to be passed between these two orchestrations. For instance, the `print boarding pass` service requires a confirmation message `check booking response` from the `process passenger information service` to proceed. We use *asynchronous* message passing to demonstrate that an individual orchestration can progress in execution while waiting for a response from a choreography partner. Blocking can take place at a different service, i.e., where the response is actually needed; or, in terms of *states*, the internal orchestration can change state and block while waiting for a return from the choreography partner.

To prevent extended deadlocks (due to absence in availability of certain services), parts of the choreography may reconfigure. In our example we fix the *passenger orchestration* (monitored with SLAs) and allow reconfiguration of the *airline orchestration* and of the message passing in the *airport check-in* choreography. The QoS analysis is then performed on the *passenger orchestration* with changes reflected in the observed outputs. Some



(a) Runtime Configuration A: Change in *airline orchestration* and choreography message passing.



(b) Runtime Configuration B: Improved services' QoS in *airline orchestration*; no change in message passing.

Figure 3. Reconfigurations of the *airport check-in* choreography.

assumptions may be made about analysis such as QoS distributions of the atomic services being analyzed at design-time. The following two configurations are studied (Fig. 3):

- *Runtime Reconfiguration A* (Fig 3(a)): The *airline orchestration* is reconfigured to produce lower latency by calling `process passenger information` and `associate baggage tag` services in parallel. It is sufficient to imagine a case where the computation of these services are performed in parallel to increase efficiency of the *airline orchestration*. This affects some of the messages that are passed within the choreography. Such reconfigurations are commonly observed in most model or agent based reconfiguration schemes [12] triggered by changes in mainly functional requirements.
- *Runtime Reconfiguration B* (Fig 3(b)): The *airline orchestration* is replaced by "better" performing services (represented by double boxes). The improvement can be in certain domains such as latency, data quality, security levels with tradeoffs of higher cost for the services. An example instance is when one of the servers of the *airline orchestration* is undergoing maintenance and is replaced by a backup service at higher cost. Such reconfigurations are typically observed in measurement-based reconfigurations as well as late-binding/substitution [2] in QoS dependent composite services.

B. QoS Analysis.

In order to analyze the effects of such reconfigurations (Fig. 3 with respect to Fig. 2) on the domains of QoS introduced in Section II, we make use of Monte-Carlo techniques used in [4], [14] to analyze the end-to-end QoS of the compositions. The composition of QoS is performed according to the rules specified in Table II.

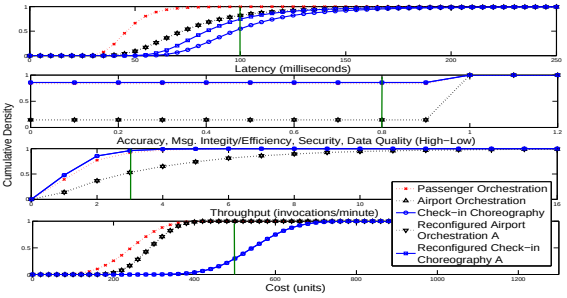
QoS values being random variables, the distributions used in the simulations include the following:

- Latency: *Student-t* (`nctrnd` in MATLAB). The values of latency are represented as random values that can be drawn from a heavy tailed distribution.
- Accuracy, Security, Message Integrity/Efficiency, Data Quality: *Binomial* (`binornd` in MATLAB). As these values are typically domains such as {valid, invalid} or {high, low}, they are accurately represented by binomial random variables.
- Throughput: *Exponential* (`exprnd` in MATLAB). Throughput/query arrival rate is represented as an exponentially increasing random variable.
- Cost: *Uniform* (`unifrnd` in MATLAB). Units of cost are drawn from a uniform distribution to be aggregated.

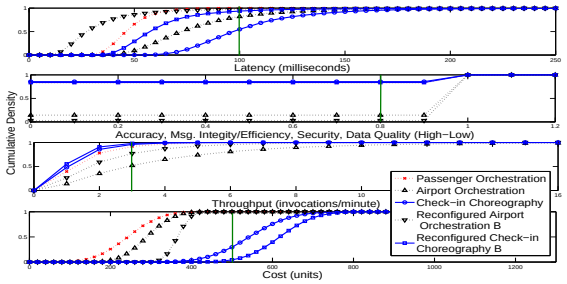
Random variables are drawn from these distributions by changing the mean and shape to model different service behavior and are composed with the algebraic rules presented in Section II with 20,000 Monte-Carlo runs in MATLAB. The effects of the reconfigurations from Section VI-A on the end-to-end QoS distributions are presented as follows:

- *Reconfiguration A* (Fig. 4(a)): As specified, this reconfiguration allows the *airline* orchestration and the message passing order to change. Though considerable the improvement in latency is not directly seen for the orchestrations in Fig. 4(a), the *airport check-in* choreography improves considerably due to changes in the message passing. As seen in Fig. 4(a), the cumulative density of choreography runs completing before 100 ms. improves from 0.5 to 0.7.
- *Reconfiguration B* (Fig. 4(b)): This reconfiguration improves certain domains for the *airline* orchestration such as lower latency/throughput and better accuracy/security/message integrity at the expense of higher costs. Similar changes are also observed for the *airport check-in* choreography. For instance in Fig. 4(b), the cumulative density of choreography runs completing before 100 ms. improves from 0.5 to 0.9. However, this is traded-off with the cumulative density of cost units < 500 reducing from 0.25 to nearly 0.

Essentially, for each QoS domain, there may be "good" and "bad" reconfigurations with respect to the current one. For the overall choreography QoS, a configuration that makes the $\oplus$ operator tend to the infimum $\bigwedge$ of the domain is "good" (e.g. lowest costing service); that which drives it towards the supremum is "bad" (e.g. worst case latency). Though these are generic observations that may be violated in data-dependent choreographies (that are bound to follow a message sequence pattern), it provides basis for automated configuration analysis. These tradeoffs can be combined with the approach provided in Section V to automate the QoS optimal selection.



(a) Runtime QoS changes due to Reconfiguration A



(b) Runtime QoS changes due to Reconfiguration B

Figure 4. QoS due to Choreography Reconfigurations.

VII. Related Work

QoS issues in web services span multiple topics such as optimal late-binding (discovery, selection, substitution) and contract management (SLAs, negotiation, monitoring). In [13], Jaeger et al. provide a comprehensive analysis of QoS composition with associated workflow patterns. Such aggregation techniques are used by Alrifai and Risse [17] and Zeng et al. [2] for optimal decomposition of global QoS constraints into local constraints. In [16], the optimization of dynamic service compositions are modeled as a multidimension-multichoice knapsack problem (MMKP). The probabilistic contractual frameworks introduced by Rosario et al. [4] has been used by Kattepur et al. for both accelerated contract composition [14] and negotiation [21] policies.

While QoS issues in composite services based on centralized control (orchestrations) has received some attention [1], the metrics relevant to choreographies are still poorly understood. This is more complex due to extensive data dependencies and message passing among participants in choreographies. In [10], Mancioppi et al. provide a structured overview of the possible metrics to be incorporated. Xiangpeng et al. [22] provide a formal analysis of integrating QoS into the language *Chor* and provide composition techniques for latency and cost. In [23], [20], stochastic models are used as an intermediate to evaluate QoS/SLAs in choreographies. Automatic triggers and strategies for QoS aware re-planning of composite orchestrations are studied in [24], which may prove promising for extension into choreographies. In our paper, we provide an algebraic formulation based on multi-dimensional probabilistic QoS models that may be composed in the case of choreographies.

Adaptive and self-healing choreographies have been studied with the survey by Leonardo et al. [12] providing a systematic overview of model, measurement, agent and formal methods driven techniques for adaptation. In [25], runtime adaptation to dynamically changing resources is studied, with optimal selection algorithms presented. Both functional and QoS requirements are studied with goal-modeling techniques combined with constraint programming approaches in [26]. The MAP protocols in [6] allow runtime distribution of the language to peers that may be specified and executed. In [27], QoS analysis and adaptation for realtime applications such as streaming services are analyzed. In [15], Ma et al. consider large compositional spaces for analysis of Pareto-optimal sets, that satisfy QoS requirements of reconfigurable compositions.

Our work combines reconfigurable choreographies along with QoS composition algebra to provide a framework for evaluating configurations. We are able to handle both orchestration and choreography QoS composition within a unified framework. Building on the choreography architectural model provided in [11], we extend the functional specifications to integrate QoS increments. Evaluation of improvements/deterioration in QoS domains is further achieved through multi-objective analysis such as goal programming [19]. Such a framework could prove critical to understand the effect of functional changes (reconfiguration) and corresponding QoS composition in the case of service choreographies.

VIII. Conclusions

While QoS composition and resulting optimal late binding have received considerable attention in case of web services orchestrations, this is still not well understood for choreographies. The problem is punctuated by intricate links with functionality, dynamic configurations and multi-dimensional QoS metrics in choreographies. In this paper, we firstly provide an algebra to compose QoS metrics from choreographies that have links to functional aspects. A corresponding framework is provided to examine QoS increments within choreography specifications of the *CHOReOS* project. The metrics are then used to analyze the effects of runtime reconfigurations on the end-to-end QoS and consequent tradeoffs. This can lead to optimal choreography configurations that may be selected with goal programming methods, for instance.

Future work would involve automatic *triggers*: self-adapting configurations could monitor deterioration or goal deviations and trigger reconfiguration.

References

[1] J. O. Sullivan, D. Edmond, and A. T. Hofstede, "What's in a service? Towards accurate description of non-functional service properties," *Distributed and Parallel Databases*, vol. 12, pp. 117–133, 2002.

[2] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. on Software Engineering*, vol. 30, pp. 311–326, 2004.

[3] P. Bhoj, S. Singhal, and S. Chutani, "SLA management in federated environments," *Computer Networks*, vol. 35(1), pp. 5–24, 2001.

[4] S. Rosario, A. Benveniste, and C. Jard, "Flexible probabilistic QoS management of transaction based web services orchestrations," in *ICWS*, 2009, pp. 107 –114.

[5] OASIS, "Web services business process execution language version 2.0," OASIS Web Services Business Process Execution Language, Tech. Rep., 2007.

[6] A. Barker, C. D. Walton, and D. Robertson, "Choreographing web services," *IEEE Trans. on Services Computing*, vol. 2, pp. 152–166, 2009.

[7] S. A. White, "Process modeling notations and workflow patterns," IBM, Tech. Rep., 2009.

[8] W3C, "Web services choreography description language version 1.0," W3C working draft, Tech. Rep., 2004.

[9] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending bpel for modeling choreographies," in *ICWS*, 2007.

[10] M. Mancioppi, M. Perepletchikov, C. Ryan, W.-J. van den Heuvel, and M. P. Papazoglou, "Towards a quality model for choreography," in *ICSOC*, 2010.

[11] CHOReOS, "Deliverable D1.3: Initial architectural style for CHOReOS choreographies," Large Scale Choreographies for the Future Internet, Tech. Rep., 2011.

[12] L. A. F. Leite, G. A. Oliva, G. M. Nogueira, M. A. Gerosa, F. Kon, and D. S. Milojicic, "A systematic literature review of service choreography adaptation," *Service Oriented Computing and Applications*, pp. 1–18, 2012.

[13] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS aggregation for web service composition using workflow patterns," in *Intl. Enterprise Distributed Object Computing Conf.*, 2004.

[14] A. Kattepur, "Importance sampling of probabilistic contracts in web services," in *ICSOC*, 2011.

[15] H. Ma, F. Bastani, I.-L. Yen, and H. Mei, "QoS-driven service composition with reconfigurable services," *IEEE Trans. on Services Computing*, vol. 6, no. 1, pp. 20–34, 2013.

[16] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, vol. 1, 2007.

[17] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *WWW Conf.*, 2009.

[18] A. Kattepur, A. Benveniste, and C. Jard, "Optimizing decisions in web services orchestrations," in *ICSOC*, 2011.

[19] D. F. Jones and M. Tamiz, *Practical Goal Programming*. Springer Books, 2010.

[20] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A stochastic programming approach for QoS-aware service composition," in *8th Intl. Symp. on Cluster Computing and the Grid*, 2008.

[21] A. Kattepur, A. Benveniste, and C. Jard, "Negotiation strategies for probabilistic contracts in web services orchestrations," in *ICWS*, 2012.

[22] Z. Xiangpeng, C. Chao, Y. Hongli, and Q. Zongyan, "A QoS view of web service choreography," in *Intl. Conf. on e-Business Engineering*, 2007.

[23] A. P. Diaz and D. M. Batista, "A methodology to define QoS and SLA requirements in service choreographies," in *Intl. Wksp. on Computer Aided Modeling and Design of Communication Links and Networks*, 2012.

[24] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-aware replanning of composite web services," in *ICWS*, 2005.

[25] V. Poladian, J. P. Sousa, D. Garlan, and M. Shaw, "Dynamic configuration of resource-aware services," in *ICSE*, 2004.

[26] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, "Using constraint programming to manage configurations in self-adaptive systems," *IEEE Computer*, vol. 12, pp. 48–55, 2012.

[27] F. Buccafurri, P. D. Meo, M. Fugini, R. Furnari, A. Goy, G. Lax, P. Lops, S. Modafferi, B. Pernici, D. Redavid, G. Semeraro, and D. Ursino, "Analysis of QoS in cooperative services for real time applications," *Data & Knowledge Engineering*, vol. 67, pp. 463–484, 2008.